

# **CS425 Final Project\_Team-T\_Inventory Management System**

S.No	Name	A-number	Contribution(%)
1.	Hemanth Thathireddy	A20525346	20%
2.	Satya Dineswara Reddy Satti	A20528639	20%
3.	Sai Manasa Basani	A20544372	20%
4.	Alavalapati Veera manohar Reddy	A20526070	20%
5.	Ying Zhang	A20547843	20%

**3rd Deliverable** (Test a variety of SQL queries)

## **1) Basic Select Query:**

*Query:*

```
SELECT * FROM Product;
```

*Explanation:* This query retrieves all columns from the Product table, displaying details of all available products.

**Output/result:**

The screenshot shows a database management interface with the following details:

- Schemas:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Views:** CustomerInfoVieww, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsViewi, ShipmentDetailView
- Stored Procedures:**
- Object Info:** Invoice
- Table:** Invoice
- Columns:**
  - InvoiceID:** int PK
  - OrderID:** int
  - InvoiceDate:** date
  - CID:** int
  - PaymentInfo:** varchar(255)
- Result Grid:** Shows 20 rows of product data with columns: PID, PName, SKU, Quantity, Price, Location, CategoryID.
- Action Output:** Shows the query "SELECT \* FROM Product;" and the result "20 row(s) returned".
- Execution Plan:** Shows "0.214 sec / 0.0015 sec".

## 2) Filtering with WHERE Clause:

*Query:*

```
SELECT PName, Price FROM Product WHERE Price < 50;
```

*Explanation:* This query retrieves the names and prices of products with a price less than 50.

**Output/result:**

The screenshot shows a database interface with the following details:

- Schemas:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Views:** CustomerInfoVieww, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsViewi, ShipmentDetailView
- Stored Procedures:**
- Object Info:** Table: Invoice
- Columns:**
  - InvoiceID:** int PK
  - OrderID:** int
  - InvoiceDate:** date
  - CID:** int
  - PaymentInfo:** varchar(255)

**Query Results:**

```

609
610
611
612
613
614
615 •  SELECT PName, Price FROM Product WHERE Price < 50;
616
617
100% 1:609

```

PName	Price
T-shirt	19.99
Jeans	39.99
Dress	49.99
Novel	9.99
Cookbook	14.99
Makeup Kit	29.99
Perfume	49.99
Dumbbells	29.99
Teddy Bear	14.99
Protein Powder	34.99

**Action Output:**

Action	Time	Response	Duration / Fetch Time
SELECT PName, Price FR...	196 11:07:23	10 row(s) returned	0.0071 sec / 0.00001...

### 3) Aggregate Function - Total Orders Amount:

*Query:*

```
SELECT SUM(TotalAmount) AS TotalOrdersAmount FROM orderr;
```

*Explanation:* This query calculates the total amount of all orders in the 'orderr' table.

**Output/result:**

```

611
612
613
614 •  SELECT SUM(TotalAmount) AS TotalOrdersAmount FROM orderr;
615
616
617
618
619

```

TotalOrdersAmount
8778.92

Result 47

Action Output

	Time	Action	Response	Duration / Fetch Time
✓ 197	11:08:28	SELECT SUM(TotalAmou...	1 row(s) returned	0.337 sec / 0.000024...

## 4) Joining Tables - Customer and Order:

Query:

```

SELECT CName, OrderDate FROM Customer
JOIN orderr ON Customer.CID = orderr.CID;

```

Explanation: This query joins Customer and orderr tables and retrieves customer names and corresponding order dates.

**Output/result:**

The screenshot shows a database interface with the following details:

- Schemas:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Selected Table:** Invoice
- Query Results:**

```

613
614
615
616 •  SELECT CName, OrderDate FROM Customer
617     JOIN orderr ON Customer.CID = orderr.CID;
618
619
620
621
  
```

CName	OrderDate
John Doe	2023-10-01
Jane Smith	2023-10-02
Michael Johnson	2023-10-03
Emily Davis	2023-10-04
David Wilson	2023-10-05
Emma Garcia	2023-10-06
Daniel Martinez	2023-10-07
Olivia Robinson	2023-10-08
Liam Clark	2023-10-09
Sophia Hall	2023-10-10
Ethan Lewis	2023-10-11
Ava Lee	2023-10-12
Mia Young	2023-10-13
Noah Adams	2023-10-14
Isabella Turner	2023-10-15
Lucas Hall	2023-10-16
Amelia Ward	2023-10-17
Aiden Scott	2023-10-18
Charlotte Lewis	2023-10-19
Henry Turner	2023-10-20
- Object Info:** Table: Invoice
- Columns:**
  - InvoiceID**: int PK
  - OrderID**: int
  - InvoiceDate**: date
  - CID**: int
  - PaymentInfo**: varchar(255)
- Action Output:**

	Time	Action	Response	Duration / Fetch Time
✓ 198	11:10:14	SELECT CName, OrderDa...	20 row(s) returned	0.015 sec / 0.000011...

## 5) : Number of products in each category

Query:

```

SELECT CategoryName, COUNT(*) AS ProductCount FROM ProductCategory
JOIN Product ON ProductCategory.CategoryID = Product.CategoryID
GROUP BY CategoryName;
  
```

Explanation: This query counts the number of products in each category and displays the category name along with the product count.

**Output/result:**

The screenshot shows a database interface with the following details:

- Schemas:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Views:** CustomerInfoVieww, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsViewi, ShipmentDetailView
- Stored Procedures:** None
- Object Info:** Table: Invoice
- Columns:**
  - InvoiceID:** int PK
  - OrderID:** int
  - InvoiceDate:** date
  - CID:** int
  - PaymentInfo:** varchar(255)
- Query:**

```

614
615
616
617 •  SELECT CategoryName, COUNT(*) AS ProductCount FROM ProductCategory
618 JOIN Product ON ProductCategory.CategoryID = Product.CategoryID
619 GROUP BY CategoryName;
620
621
622
  
```
- Result Grid:**

CategoryName	ProductCount
Electronics	3
Clothing	3
Home and Living	3
Books	2
Beauty	2
Sports	2
Toys	1
Automotive	1
Jewelry	1
Furniture	1
Health and Fitness	1
- Action Output:**

	Time	Action	Response	Duration / Fetch Time
✓ 199	11:11:21	SELECT CategoryName,...	11 row(s) returned	0.030 sec / 0.000013...

## 6) Window Function - Ranking Products by Price:

Query:

```

SELECT PName, Price,
       RANK() OVER (ORDER BY Price DESC) AS rankk
  FROM Product;
  
```

Explanation: This query assigns a rank to products based on their prices, ordering them in descending order.

**Output/result:**

Table: **Invoice**

**Columns:**

- InvoiceID** int PK
- OrderID** int
- InvoiceDate** date
- CID** int
- PaymentInfo** varchar(255)

PName	Price	rankk
Diamond Ring	999.99	1
Laptop	799.99	2
Bed	799.99	2
Smartphone	499.99	4
Sofa	499.99	4
Tablet	299.99	6
Coffee Table	149.99	7
Office Chair	149.99	7
Car Battery	99.99	9
Running Shoes	79.99	10
Dress	49.99	11
Perfume	49.99	11
Jeans	39.99	13
Protein Powder	34.99	14
Makeup Kit	29.99	15
Dumbbells	29.99	15
T-shirt	19.99	17
Cookbook	14.99	18
Teddy Bear	14.99	18
Novel	9.99	20

Action Output

Time	Action	Response	Duration / Fetch Time
200 11:12:22	SELECT PName, Price,...	20 row(s) returned	0.014 sec / 0.000011...

## 7) Calculating Running Total of Quantity Sold:

```
SELECT PName, Quantity,
       SUM(Quantity) OVER (ORDER BY PID) AS RunningTotalQuantity
  FROM Product
 ORDER BY PID;
```

Explanation: This query calculates the running total of quantities sold for products, ordered by their Product IDs.

**Output/result:**

```

626
627
628 •  SELECT PName, Quantity,
629          SUM(Quantity) OVER (ORDER BY PID) AS RunningTotalQuantity
630      FROM Product
631      ORDER BY PID;
632
633
634

```

PName	Quantity	RunningTotalQuant...
Laptop	50	50
Smartphone	100	150
Tablet	30	180
T-shirt	200	380
Jeans	150	530
Dress	80	610
Sofa	20	630
Coffee Table	40	670
Bed	15	685
Novel	300	985
Cookbook	120	1105
Makeup Kit	80	1185
Perfume	150	1335
Running S...	90	1425
Dumbbells	50	1475
Teddy Bear	200	1675
Car Battery	30	1705
Diamond R...	10	1715
Office Chair	25	1740
Protein Po...	100	1840

Result 52

Action Output

Action	Time	Response	Duration / Fetch Time
205	12:04:50	SELECT PName, Quantity...	20 row(s) returned 0.010 sec / 0.000012...

## 8) OLAP Function - Running Total of Orders:

Query:

```

SELECT OrderID, OrderDate, TotalAmount,
       SUM(TotalAmount) OVER (ORDER BY OrderDate) AS RunningTotal
FROM orderr;

```

Explanation: This query calculates the running total of orders over time, ordering them by order date.

**Output/result:**

SCHEMAS

InventoryManagementDB

Tables

- > Customer
- > **Invoice**
- > OrderProduct
- > orderr
- > Product
- > ProductCategory
- > Shipment

Views

- > CustomerInfoVieww
- > OrderSummaryVie
- > ProductCategorySummaryView
- > ProductDetailsViewi
- > ShipmentDetailView

Stored Procedures

Object Info Session

**Table: Invoice**

**Columns:**

<b>InvoiceID</b>	int PK
<b>OrderID</b>	int
<b>InvoiceDate</b>	date
<b>CID</b>	int
<b>PaymentInfo</b>	varchar(255)

```

631 ORDER BY PID;
632
633
634 •   SELECT OrderID, OrderDate, TotalAmount,
635           SUM(TotalAmount) OVER (ORDER BY OrderDate) AS RunningTotal
636   FROM orderr;
637
638
639

```

100% 2:639

Result Grid Filter Rows: Export: Result Grid

OrderID	OrderDate	TotalAmount	RunningTotal
10001	2023-10-01	299.99	299.99
10002	2023-10-02	459.97	759.96
10003	2023-10-03	799.95	1559.91
10004	2023-10-04	199.99	1759.90
10005	2023-10-05	679.96	2439.86
10006	2023-10-06	119.99	2559.85
10007	2023-10-07	589.93	3149.78
10008	2023-10-08	899.92	4049.70
10009	2023-10-09	329.99	4379.69
10010	2023-10-10	179.98	4559.67
10011	2023-10-11	239.97	4799.64
10012	2023-10-12	409.96	5209.60
10013	2023-10-13	729.95	5939.55
10014	2023-10-14	279.94	6219.49
10015	2023-10-15	149.93	6369.42
10016	2023-10-16	609.92	6979.34
10017	2023-10-17	499.91	7479.25
10018	2023-10-18	339.90	7819.15
10019	2023-10-19	169.89	7989.04
10020	2023-10-20	789.88	8778.92

Result 53 Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
206 12:06:17	SELECT OrderID, OrderD...	20 row(s) returned	0.032 sec / 0.000060...

## 9) Top N Customers by Total Amount Spent:

```

SELECT o.CID, c.CName, SUM(o.TotalAmount) AS TotalSpent
FROM orderr o
JOIN Customer c ON o.CID = c.CID
GROUP BY o.CID, c.CName
ORDER BY TotalSpent DESC
LIMIT 5;

```

Explanation: This query identifies the top 5 customers who have spent the most based on the total amount in their orders.

**Output/result:**

```

632
633
634 •   SELECT o.CID, c.CName, SUM(o.TotalAmount) AS TotalSpent
635     FROM orderr o
636     JOIN Customer c ON o.CID = c.CID
637     GROUP BY o.CID, c.CName
638     ORDER BY TotalSpent DESC
639     LIMIT 5;
640

```

CID	CName	TotalSpent
108	Olivia Robinson	899.92
103	Michael Johnson	799.95
120	Henry Turner	789.88
113	Mia Young	729.95
105	David Wilson	679.96

Result 54 | Read Only

Action Output | Time | Action | Response | Duration / Fetch Time

211 12:12:25 SELECT o.CID, c.CName,... 5 row(s) returned 0.0077 sec / 0.00001...

## 10) Total Revenue by Month:

```

SELECT EXTRACT(MONTH FROM OrderDate) AS Month,
       EXTRACT(YEAR FROM OrderDate) AS Year,
       SUM(TotalAmount) AS MonthlyRevenue
  FROM orderr
 GROUP BY EXTRACT(MONTH FROM OrderDate), EXTRACT(YEAR FROM OrderDate)
 ORDER BY Year, Month;

```

Explanation: This query calculates the total revenue for each month and year, providing a monthly breakdown of sales.

**Output/result:**

The screenshot shows a database management interface with the following details:

- Schemas:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Views:** CustomerInfoVieww, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsViewi, ShipmentDetailView
- Stored Procedures:** None
- Object Info:** Table: orderr
- Columns:**
  - OrderID:** int PK
  - CID:** int
  - OrderDate:** date
  - TotalAmount:** decimal(10,2)
- Result Grid:** Shows a single row for October 2023 with MonthlyRevenue: 8778.92.
- Action Output:** Shows the executed query: SELECT EXTRACT(MONTH FROM OrderDate) AS Month, EXTRACT(YEAR FROM OrderDate) AS Year, SUM(TotalAmount) AS MonthlyRevenue FROM orderr GROUP BY EXTRACT(MONTH FROM OrderDate), EXTRACT(YEAR FROM OrderDate) ORDER BY Year, Month;
- Execution Plan:** Shows the execution time: 0.017 sec / 0.000016...

## 11) Subquery - Products Below Average Price:

Query:

```
SELECT PName, Price
FROM Product
WHERE Price < (SELECT AVG(Price) FROM Product);
```

Explanation: This query finds products with prices below the average price of all products in the Product table.

**Output/result:**

```

646
647
648
649 •  SELECT PName, Price
650   FROM Product
651   WHERE Price < (SELECT AVG(Price) FROM Product);
652
653
654

```

PName	Price
T-shirt	19.99
Jeans	39.99
Dress	49.99
Coffee Table	149.99
Novel	9.99
Cookbook	14.99
Makeup Kit	29.99
Perfume	49.99
Running Shoes	79.99
Dumbbells	29.99
Teddy Bear	14.99
Car Battery	99.99
Office Chair	149.99
Protein Powder	34.99

Product 56      Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
213 12:15:34	SELECT PName, Price F...	14 row(s) returned	0.035 sec / 0.000011...

## 12) CASE Statement - Categorizing Products:

Query:

```

SELECT PName,
CASE
    WHEN Price < 50 THEN 'Low Range'
    WHEN Price >= 50 AND Price < 200 THEN 'Mid Range'
    ELSE 'High Range'
END AS PriceCategory
FROM Product;

```

Explanation: This query categorizes products into price ranges using a CASE statement.

**Output/result:**

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree view is open, showing the 'InventoryManagementDB' schema with its tables (Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment), views (CustomerInfoVieww, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsViewi, ShipmentDetailView), and stored procedures. The 'Customer' table is selected.

In the main area, a query is being run:

```

653
654 •   SELECT PName,
655     CASE
656       WHEN Price < 50 THEN 'Low Range'
657       WHEN Price >= 50 AND Price < 200 THEN 'Mid Range'
658       ELSE 'High Range'
659     END AS PriceCategory
660   FROM Product;
661

```

The results are displayed in a grid:

PName	PriceCategory
Laptop	High Range
Smartphone	High Range
Tablet	High Range
T-shirt	Low Range
Jeans	Low Range
Dress	Low Range
Sofa	High Range
Coffee Table	Mid Range
Bed	High Range
Novel	Low Range
Cookbook	Low Range
Makeup Kit	Low Range
Perfume	Low Range
Running S...	Mid Range
Dumbbells	Low Range
Teddy Bear	Low Range
Car Battery	Mid Range
Diamond R...	High Range
Office Chair	Mid Range
Protein Po...	Low Range

Below the grid, it says 'Result 58' and 'Read Only'. At the bottom, there's an 'Action Output' section with a table showing the execution details:

Action	Time	Action	Response	Duration / Fetch Time
216	12:18:26	SELECT PName,	CA... 20 row(s) returned	0.0014 sec / 0.00001...

### 13) DELETE Statement - Remove Low-Quantity Products:

Query:

```
DELETE FROM Product WHERE Quantity < 50;
```

Explanation: This query deletes products with quantities less than 50 from the Product table.

Output/result:

## Before:

	PID	PName	SKU	Quantity	Price	Location	CategoryID
1	1	Laptop	LT1001	50	799.99	Warehouse A	1001
2	2	Smartphone	SP2002	100	499.99	Store 1	1001
3	3	Tablet	TB3003	30	299.99	Store 2	1001
4	4	T-shirt	TS4004	200	19.99	Warehouse B	1002
5	5	Jeans	JN5005	150	39.99	Store 1	1002
6	6	Dress	DR6006	80	49.99	Store 3	1002
7	7	Sofa	SF7007	20	499.99	Warehouse C	1003
8	8	Coffee Table	CT8008	40	149.99	Store 2	1003
9	9	Bed	BD9009	15	799.99	Warehouse A	1003
10	10	Novel	NV10010	300	9.99	Store 4	1004
11	11	Cookbook	CB11011	120	14.99	Store 3	1004
12	12	Makeup Kit	MK12012	80	29.99	Store 5	1005
13	13	Perfume	PF13013	150	49.99	Store 6	1005
14	14	Running Shoes	RS14014	90	79.99	Store 7	1006
15	15	Dumbbells	DB15015	50	29.99	Store 8	1006
16	16	Teddy Bear	TB16016	200	14.99	Store 9	1007
17	17	Car Battery	CB17017	30	99.99	Store 10	1008
18	18	Diamond Ring	DR18018	10	999.99	Store 11	1009
19	19	Office Chair	OC19019	25	149.99	Store 12	1010
20	20	Protein Powder	PP20020	100	34.99	Store 13	1011

## After:

	PID	PName	SKU	Quantity	Price	Location	CategoryID
	1	Laptop	LT1001	50	799.99	Warehouse A	1001
	2	Smartphone	SP2002	100	499.99	Store 1	1001
	4	T-shirt	TS4004	200	19.99	Warehouse B	1002
	5	Jeans	JN5005	150	39.99	Store 1	1002
	6	Dress	DR6006	80	49.99	Store 3	1002
	10	Novel	NV10010	300	9.99	Store 4	1004
	11	Cookbook	CB11011	120	14.99	Store 3	1004
	12	Makeup Kit	MK12012	80	29.99	Store 5	1005
	13	Perfume	PF13013	150	49.99	Store 6	1005
	14	Running Shoes	RS14014	90	79.99	Store 7	1006
	15	Dumbbells	DB15015	50	29.99	Store 8	1006
	16	Teddy Bear	TB16016	200	14.99	Store 9	1007
	20	Protein Powder	PP20020	100	34.99	Store 13	1011

## 14) UPDATING Records - Change Product Location:

Query:

```
UPDATE Product SET Location = 'Warehouse D' WHERE CategoryID = 1003;
```

Explanation: This query updates the location of products in the 'Home and Living' category to 'Warehouse D'.

## Output/result:

The screenshot shows a database management interface with the following details:

- SCHEMAS:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product, ProductCategory, Shipment
- Views:** CustomerInfoViewvw, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsView, ShipmentDetailView
- Stored Procedures:** None
- Object Info:** Table: Product
- Columns:**
  - PID: int PK
  - PName: varchar(255)
  - SKU: varchar(50)
  - Quantity: int
  - Price: decimal(10,2)
  - Location: varchar(255)
  - CategoryID: int
- Result Grid:** Shows 20 rows of product data. The last row (index 20) is highlighted.
- Action Output:** Shows the execution details:

	Time	Action	Response	Duration / Fetch Time
218	12:21:55	select * from Product LIM...	20 row(s) returned	0.00083 sec / 0.000...

## 15) Group By with Having Clause - Categories with Expensive Products:

Query:

```
SELECT ProductCategory.CategoryName, AVG(Product.Price) AS AveragePrice
FROM Product
JOIN ProductCategory ON Product.CategoryID = ProductCategory.CategoryID
GROUP BY ProductCategory.CategoryName
HAVING AVG(Product.Price) > 200;
```

Explanation: This query finds categories with an average product price greater than 200, grouping the results by category name.

## Output/result:

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree is expanded to show the 'InventoryManagementDB'. Under 'Tables', the 'Product' table is selected. The main pane displays the following SQL query:

```

668
669
670 •  SELECT ProductCategory.CategoryName, AVG(Product.Price) AS AveragePrice
671   FROM Product
672   JOIN ProductCategory ON Product.CategoryID = ProductCategory.CategoryID
673   GROUP BY ProductCategory.CategoryName
674   HAVING AVG(Product.Price) > 200;
675
676

```

The result grid shows the following data:

CategoryName	AveragePrice
Electronics	533.323333
Home and Living	483.323333
Jewelry	999.990000

At the bottom, the status bar indicates 'Result 60' and the execution details: '219 12:26:38 SELECT ProductCategory... 3 row(s) returned 0.017 sec / 0.000069...'.

## 16) Nested Subquery - Customers Who Made Multiple Orders:

Query:

```

SELECT CName
FROM Customer
WHERE CID IN (SELECT CID FROM orderr GROUP BY CID HAVING
COUNT(*) > 1);

```

Explanation: This query finds customers who have made more than one order by using a nested subquery.

## Output/result:

The screenshot shows a database interface with the following details:

- SCHEMAS:** InventoryManagementDB
- Tables:** Customer, Invoice, OrderProduct, orderr, Product (selected), ProductCategory, Shipment
- Views:** CustomerInfoViewvw, OrderSummaryVie, ProductCategorySummaryView, ProductDetailsView, ShipmentDetailView
- Stored Procedures:** None
- Object Info:** Table: Product
- Columns:**
  - PID (int PK)
  - PName (varchar(255))
  - SKU (varchar(50))
  - Quantity (int)
  - Price (decimal(10,2))
  - Location (varchar(255))
  - CategoryID (int)
- Result Grid:** CName (Customer) | PID | PName | SKU | Quantity | Price | Location | CategoryID
- Action Output:** Time: 220, Action: SELECT CName FROM C..., Response: 0 row(s) returned, Duration / Fetch Time: 0.0094 sec / 0.0000...

## 17) Cross Join - Cartesian Product of Customers and Products:

Query:

```
SELECT CName, PName
FROM Customer
CROSS JOIN Product;
```

Explanation: This query performs a cross join between Customer and Product tables, creating a Cartesian product of customers and products.

## Output/result:

**SCHEMAS**

Filter objects

**InventoryManagementDB**

- Tables
  - Customer
  - Invoice
  - OrderProduct
  - orderr
  - Product**
  - ProductCategory
  - Shipment
- Views
  - CustomerInfoVieww
  - OrderSummaryVie
  - ProductCategorySummaryView
  - ProductDetailsViewi
  - ShipmentDetailView
- Stored Procedures

Object Info Session

**Table: Product**

**Columns:**

<b>PID</b>	int PK
<b>PName</b>	varchar(255)
<b>SKU</b>	varchar(50)
<b>Quantity</b>	int
<b>Price</b>	decimal(10,2)
<b>Location</b>	varchar(255)
<b>CategoryID</b>	int

```

681
682 •   SELECT CName, PName
683   FROM Customer
684   CROSS JOIN Product;
685
100% 20:684

```

Result Grid Filter Rows: Search Export:

CName	PName
Henry Turner	Laptop
Charlotte Lewis	Laptop
Aiden Scott	Laptop
Amelia Ward	Laptop
Lucas Hall	Laptop
Isabella Turner	Laptop
Noah Adams	Laptop
Mia Young	Laptop
Ava Lee	Laptop
Ethan Lewis	Laptop
Sophia Hall	Laptop
Liam Clark	Laptop
Olivia Robinson	Laptop
Daniel Martinez	Laptop
Emma Garcia	Laptop
David Wilson	Laptop
Emily Davis	Laptop
Michael John...	Laptop
Jane Smith	Laptop
John Doe	Laptop
Henry Turner	Smart...
Charlotte Lewis	Smart...
Aiden Scott	Smart...
Amelia Ward	Smart...
Lucas Hall	Smart...
Isabella Turner	Smart...
Noah Adams	Smart...

Result 62 Read Only

Action Output

Time	Action	Response	Duration / Fetch Time
221 12:28:47	SELECT CName, PName...	400 row(s) returned	0.0075 sec / 0.00004...

Result Grid Form Editor Field Types Query Stats Execution Plan